MQCC: Maximum Queue Congestion Control for Multipath Networks with Blockage

Scott Pudlewski, Brooke Shrader, Laura Herrera, Nathaniel M. Jones, Andrew P. Worthen
Massachusetts Institute of Technology
Lincoln Laboratory
e-mail: {scott.pudlewski, brooke.shrader}@ll.mit.edu

***Abstract*—This paper presents a transport layer protocol for multi-path networks with blockage. Using urban SATCOM as an example, we see from data taken from a 2006 measurement campaign that these blockages are generally on the order of $1-5$ seconds in length and the links are blocked approximately 33% of the time. To compensate for this type of impairment, we have developed a multipath IP overlay routing algorithm, a random linear coding reliability scheme, and a maximum-queue-based (MQCC) congestion control algorithm.**

MQCC uses average buffer occupancy as a measure of the congestion in a network (as opposed to packet loss or round trip time (RTT)) and updates the transmission rate of each source to avoid network congestion. This allows us to design a congestion control algorithm that is independent of the channel conditions and can be made resilient to channel losses. The reliability scheme uses selective negative acknowledgments (SNACKs) to guarantee packet delivery to the destination. We show through simulation that we can approach the optimal benchmark in realistic lossy blockage channels.

I. Introduction

Many high frequency wireless links experience blockage due to physical obstructions. One common example of this is mobile satellite-terrestrial links, which can be modeled using a Markov process as an on-off blockage channel [2]. In this example, the blockage is due to the terrestrial nodes moving near and around obstructions that intermittently block the line-of-sight link to the satellite. This causes blockages in the channel that vary in time based on the speed on the vehicle and the environment. For example, for a vehicle moving in an urban environment, the passing buildings can cause blockages on the order of seconds in length [2].

Especially challenging are recurring link blockages that last on the order of seconds. Forward error correction schemes used to combat small-scale fading would impose large delays when applied to longer outages. On the other hand, persistent outages lasting minutes or longer are handled by current routing protocols. This includes routing protocols developed for the delay/disruption-tolerant networking paradigm, where a communication link between a pair of nodes only exists occasionally, as governed by the node pairs' inter-contact time. By contrast, in our scenario, communication links between pairs of nodes are persistent, but the link is sometimes blocked. Our focus is on link blockages that span seconds, as determined by the speed of the mobile and size of the obstruction. This problem is not addressed by current techniques.

We argue that the best way to combat blockages at these timescales is at and above the network layer. In [3], we demonstrate an approach consisting of three components.

- Concurrent routing over multiple paths which provides robustness to blockages on any single link or path coupled with end-to-end block coding;
- queue-length-based congestion control and a loss recovery scheme that uses selective NACKs; and
- an IP-overlay implementation.

In this work, we focus on the second item. Specifically, we develop a multipath queue-length-based congestion control algorithm designed to perform well in networks with link blockage. By taking advantage of the overlay implementation scheme described in [3], we can explicitly measure the queue length of a subset of nodes in the network. We then use these queue lengths to develop a protocol that can be shown to distributively solve a sum-network-utility maximization problem.

The remainder of the paper is structured as follows. In Section II, we discuss related work. Section III introduces the MQCC congestion control protocol, while Section IV introcudes the selective negative acknowledgment reliability protocol. We evaluate the performance in Section V, and finally conclude the paper with Section VI.

II. Related Work

Most existing transport layer congestion control protocols fall into one of three categories.

- **Loss Based Congestion Control.** The vast majority of congestion control algorithms today use packet losses as an indication of congestion. This includes TCP Reno [4], Compound TCP [5], and TCP Cubic [6], among many others. In all of these cases, the sender and receiver use an ACK based feedback mechanism to inform the sender whether a packet is or is not received at the destination. Whenever a packet is not received, the sender assumes that a queue somewhere in the network has overflowed.

Distribution A: Public Release
A short description of this work appeared at a conference demo session [1].
This work is sponsored by the Assistant Secretary of Defense for Research & Engineering under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Since queue overflow is an indication of congestion, the source node decreases the window size (or send rate) to compensate. In wired networks, since packet loss rates due to bit errors are relatively rare, this approach works very well. Unfortunately, higher error rates in wireless networks result in a great deal of "false" congestion indications, resulting in underutilization of the network [4].

- **Delay Based Congestion Control.** The time it takes a packet to traverse the network from source to destination can be used as an indication of the overall network condition [7], [4]. In TCP Vegas [7], the difference between the actual and expected (i.e., propagation) delay is used as an indication of the queue length at the bottleneck link. Indeed, in [8], it was shown that this update can be proven to converge to an optimal weighted-proportional-fair rate allocation throughout the network. The main drawback for our blockage scenario is that, because of the length of the outages, it is difficult or impossible to accurately measure any end-to-end metric.
- **Explicit Congestion Notification (ECN).** ECN based systems use explicit congestion messages between a router experiencing congestion and the sender to indicate congestion. By separating the channel loss behavior and the congestion loss behavior, a congestion control algorithm can deal with any congestion events while allowing other techniques to handle channel losses. However, for an ECN based system to work, every router that could experience congestion *must* be able to signal congestion to the source. While we can control a subset of nodes, there will still exist legacy devices that we are unable to modify.

In addition to these traditional approaches, there are a couple of newer approaches that are relevant to lossy wireless networks. Multipath TCP (MPTCP) schemes [9], [10] explore the design and implementation of multipath versions of a loss-based TCP protocol. The authors or [10] design an MPTCP protocol specifically with fairness in mind, and define two goals of the protocol.

1) The throughput of the multipath TCP flow should be at least equal to the throughput of a single-path TCP flow choosing the *best* (i.e., highest throughput) path.
2) Regardless of the number of paths chosen, the multipath flow must not take more capacity on any combination of paths than if it were a single-path TCP flow choosing the best path.

Essentially, MPTCP should always be at least as good as single-path TCP, and it should not compete with other flows any more aggressively than an equivalent "best case" single path flow would. To accomplish this, the authors introduce a loss-based congestion control algorithm that controls the window sizes for each path based in part on the *total* window size for all paths used by the connection. In this work, we look at some of the same concepts as MPTCP but in a network where we can not rely on traditional feedback metrics.

In [11], the authors introduce network coding as an error correction scheme at the transport layer. They accomplished this by implementing a new "network coding layer" below the transport layer. They then demonstrate that it is often advantageous to use received *degrees of freedom* to guarantee the delivery of packets to the destination. Since the network coding layer only accounts for lost packets, they implement the protocol with TCP Vegas [7] for congestion control.

A primary advantage of [11] is that it allows the source to trade packet loss for RTT delay in a way that is transparent to the TCP protocol. However, this benefit depends on two assumptions. First, the source needs to be able to predict the correct amount of redundancy for optimal performance. In blockage channels, this is nearly impossible to do in the short term. Second, because SATCOM blockage events are highly correlated in time, the single path error correction scheme may not perform well.

Finally, loss tolerance TCP (LT-TCP) as presented in [12] is an example of a system that attempts to "fix" TCP to work with lossy wireless networks using existing techniques. The authors have taken the view that because packet losses are unreliable as a congestion indicator, they should never be acted upon as a congestion event. Instead, they assert that explicit congestion notification (ECN) is the more appropriate response. Similar to the network coding work above, they then use forward error correction (FEC) to correct for losses in the network.

The primary approach of LT-TCP is to first predict the loss rate and implement *proactive* FEC bytes to compensate for expected bit errors. If there are still errors at the receiver the source will then send *reactive* FEC bytes to allow the destination to decode the packets. This is similar in principal to the network coding scheme in [11]. While such an approach can often work very well, it is again not appropriate for blockage networks. First, the blockage lengths are long enough that "coding over them" using traditional FEC and interleaving techniques would cause significant delays. These techniques were designed for loss durations on the order of milliseconds, where blockages are on the order of seconds. Second, it is difficult to accurately predict the loss rate in any mobile wireless network, particularly a satellite blockage network whose blockages depend on line of sight impairments.

III. Maximum Queue Congestion Controller (MQCC)

Here we present the maximum queue congestion controller (MQCC) which uses both the maximum measured queue lengths and round trip time (RTT) measurements to prevent congestion. We present each phase of the congestion controller individually, justifying why two phases are needed for realistic incremental deployment. Finally, we show that the end-to-end congestion controller can be viewed as the optimal solution to a sum-network-utility maximization problem.

A. Network Model

For this work, the focus is on mobile tactical ground vehicles, which have already been deployed with IP routers

installed. Since changing this hardware would be extremely expensive, we have designed this protocol to be able to be implemented as an IP overlay network, where a small number of key nodes have been "upgraded" to include the MQCC interface and software. These *overlay* nodes will take responsibility for forcing traffic to use specific relay nodes, allowing for (at least partially) disjoint paths. These overlay nodes also allow us to measure and control the transmission rate of the packets at key points in the network, allowing us to collect information from within the network even in paths that contain nodes (such as satellite hops) that are very challenging to upgrade.

We therefore propose a two-phase system. For end-to-end congestion control, we define a system where each overlay node reports its queue size to the source node, and the the source uses these measurements to determine the presence or absence of congestion in the network. By using explicit measurements, we can avoid the ambiguity of accounting for large numbers of missing packets. For local rate control (i.e., the capacity of a link between two overlay nodes), we use an RTT based algorithm. We define each of the components individually below, and we show that the end-to-end rate control algorithm is optimal in terms of proportional fair utility maximization.

B. Source-based Overlay Congestion Control

In MQCC, the packet transmission rate at each source is updated using the measured maximum queue length $\tilde{Q}^s$ as an indication of congestion. Specifically, we update the source rate according to

$$x^s(t+1) = \begin{cases} x^s(t) + k\dfrac{\gamma^-}{\tilde{Q}^s} & \text{if } \tilde{Q}^s < \gamma^-, \\ x^s(t) - k\dfrac{\tilde{Q}^s}{\gamma^+} & \text{if } \tilde{Q}^s > \gamma^+, \\ x^s(t) & \text{else,} \end{cases} \tag{1}$$

where $x^s(t)$ is the transmission rate as source s at time t, $0 < k \leq 1$ is a user defined constant used to regulate how fast the rate adjusts at a source, and $0 < \gamma^+$ and $0 < \gamma^-$ are constants representing the maximum and minimum number of bits that will be maintained in the sum of all queues along any single path. While we will prove that this update achieves optimal-sum-utility rate allocation in Section III-D, we can easily see that such an update also makes intuitive sense. Assuming that we want to keep between γ^+ and γ^- packets in the bottleneck queue, (1) states that we should decrease the data rate if we have more than γ^+ packets, and we should increase the data rate if we have less than γ^- packets. At steady state, assuming perfect measurement, the length of the maximum queue length in the network will be between γ^+ and γ^-.

$\tilde{Q}^s$ is defined as

$$\tilde{Q}^s = \max_{i \in \mathcal{N}(s)} Q_i \tag{2}$$

where Q_i is defined as the queue length at node i, $\mathcal{N}$ is the set of all nodes in the network and $\mathcal{N}(s)$ is the set of nodes in $\mathcal{N}$ that participate in the flow from source s. Q^s is the maximum queue used by flow s, and is found through message passing to the source.

C. Local Capacity Estimation

It is assumed in (2) that the network consists entirely of overlay nodes, and each of these nodes is able to report a queue occupancy value to the source. Realistically, the bottleneck link could be in the underlay network at a node that we can not directly measure. To compensate for this, we implement a second rate control algorithm between each two overlay neighbors in the network. We refer to this as a "capacity estimation" algorithm because the goal of this algorithm is to determine the capacity of the hyperedge between each pair of overlay nodes assuming that each node may be receiving data from- and sending data to more than one other node.

The key differences between this and the end-to-end algorithm described in (1) are:

- The local capacity estimation algorithm is run *per logical link* and does not distinguish which source (or path) any data belongs to.
- The local algorithm must *estimate* the queues along the underlay path.
- This algorithm is run independently at each overlay node where data is being transmitted using the overlay network.

The capacity of each overlay link is updated using

$$c_l(t+1) = \begin{cases} c_l(t) + \kappa & \text{if } c_l(t)(D_l(t) - d_l) < \alpha \\ c_l(t) - \kappa & \text{if } c_l(t)(D_l(t) - d_l) > \beta \\ c_l(t) & \text{else,} \end{cases} \tag{3}$$

where $c_l(t)$ is the capacity in packets per second, κ is some constant number of packets per second, $D_l(t)$ is the *measured* RTT for link l at time t, d_l is the propagation delay on link l and α and β are constants with dimension of packets. Because the nodes are mobile and the paths are constantly changing, the propagation delay d_l is periodically measured using small, high priority probe packets. The update described in (3) states that the rate c_l of link l at time $t+1$ is increased or decreased by κ packets per second based on whether $c_l(t)(D_l(t) - d_l)$ is greater than β or less than α_l. Interestingly, it can easily be shown that $c_l(t)(D_l(t) - d_l)$ is the number of backlogged packets on link l, while α and β, $\beta > \alpha$ represent some range of ideal buffer occupancy.

D. Optimality of the End-to-End Congestion Controller

To demonstrate the optimality of (1), we first need to define the network model. We represent the network as a set $\mathcal{N}$ of nodes, and the set $\mathcal{L}$ represents the set of all links in the network. We indicate the set of source nodes as $\mathcal{S}$, with $\mathcal{S} \subseteq \mathcal{N}$. We define $\mathcal{L}(s)$ as the set of links in $\mathcal{L}$ used by source s. For each link l, let $\mathcal{S}(l) = \{s \in \mathcal{S} \mid l \in \mathcal{L}(s)\}$ be the set of sources whose traffic traverses link l along at least one path. By definition, $l \in \mathcal{L}(s)$ if and only if $s \in \mathcal{S}(l)$.

We would like to find a distributed source update algorithm that will maximize some function $U(x_s)$, which is a non-decreasing utility function of the data rate x_s at source s. Similar to [8], we begin by defining an optimization problem

$$\begin{aligned} \text{Maximize} \quad & \sum_{s\in\mathcal{S}} U(x_s) \\ \text{Subject To:} \quad & \sum_{s\in\mathcal{S}(l)} x_s \le c_l \qquad \forall l \in L \\ & x_s \ge 0 \qquad\qquad \forall s \in \mathcal{S}, \end{aligned} \tag{4}$$

which states that we would like to maximize the sum of $U(x_s)$ over all sources s given a capacity constraint c_l on each link l.

Following the same formulation as [8], [13], we can then find the Lagrangian of (4), as:

$$\begin{aligned} L(x,\lambda) &= \sum_{s\in\mathcal{S}} U(x_s) - \sum_{l\in\mathcal{L}} \lambda_l \left(\sum_{s\in\mathcal{S}(l)} x_s - c_l \right) \\ &= \sum_{s\in\mathcal{S}} \left(U(x_s) - x_s \sum_{l\in\mathcal{L}(s)} \lambda_l \right) + \sum_{l\in\mathcal{L}} \lambda_l c_l, \end{aligned} \tag{5}$$

where the relaxation variable λ_l represents the *penalty* or *price* for violating the capacity constraint. The Lagrange Dual function can then be defined as

$$\begin{aligned} g(\lambda) &= \max_x L(x,\lambda) \\ &= \max_x \left[\sum_{s\in\mathcal{S}} \left(U(x_s) - x_s \sum_{l\in\mathcal{L}(s)} \lambda_l \right) + \sum_{l\in\mathcal{L}} \lambda_l c_l \right] \\ &= \sum_{s\in\mathcal{S}} \max_x \left[U(x_s) - x_s \sum_{l\in\mathcal{L}(s)} \lambda_l \right] + \sum_{l\in\mathcal{L}} \lambda_l c_l. \end{aligned} \tag{6}$$

Based on the dual formulation in (6), the optimal value of x_s will be the value that uniquely solves

$$\max_x \left[U(x_s) - x_s \sum_{l\in\mathcal{L}(s)} \lambda_l \right]. \tag{7}$$

As long as $U(x_s)$ is differentiable over all values of x_s, x_s^* is defined as the value of x_s such that

$$\frac{dU(x_s)}{dx_s} = \lambda^s, \tag{8}$$

where $\lambda^s = \sum_{l\in\mathcal{L}(s)} \lambda_l$. Finally, we note that λ^s can be interpreted as

$$\lambda^s = \frac{b^s}{c^s}, \tag{9}$$

where b^s, defined as

$$b^s = \max_{p\in\mathcal{P}(s)} \sum_{i\in\mathcal{N}(p)} Q_i^{s,p}, \tag{10}$$

is the number of packets queued along the most congested path for source s, $\mathcal{N}(p)$ is the set of nodes in path p, and $\mathcal{P}(s)$ is the set of paths used by source s. c^s is the bottleneck capacity between the source and destination for source s. For a utility function

$$\frac{\gamma}{\rho_s} \log(x_s), \tag{11}$$

which can be interpreted as weighted proportional fairness where the weights are the desired queue length (γ) divided by the ratio of the bottleneck link RTT to the end-to-end RTT for source s (ρ_s), we can now find

$$x_s^* = \frac{\gamma}{\rho_s \lambda^s} = \frac{\gamma c^s}{\rho_s b^s}, \tag{12}$$

which states that the optimal data rate is the capacity times the ratio of the number of packets we would like in the queue (γ) divided by the weighted length of the queues.

Ideally, we would be done here. however, the above analysis assumes that we know both b^s and c^s exactly. Realistically, whatever we have for b^s is a delayed estimate based on values reported by the network, and c^s is unknown. However, if we rewrite (12) as

$$\frac{x_s^*}{c^s} = \frac{\gamma}{\rho_s b^s}, \tag{13}$$

two things become very clear. First, to achieve $x_s^* = c^s$, we need $\gamma = \rho_s b^s$. Second, and perhaps more important, the relationship between x_s^* and c^s is the same as that between γ and $\rho_s b^s$.

Finally, we note that, in general, b^s is very difficult to measure accurately since it involves measurements from every node along the path. Therefore, instead of using b^s, which is the sum of all queues along a path, we use $\rho_s b_{max}^s$ which is the *maximum* queue length along a path weighted by ρ_s. The measurement is weighted by ρ_s because the local capacity update between overlay nodes, defined in (3), is weighted proportional fair [14], resulting in the additional delay factor in the rates and therefore the queue lengths. Since a path can have at most one bottleneck link [14], this will in general give an answer that is close to $\rho_s b^s$.

IV. Selective Negative Acknowledgment (SNACK) Based Reliability

Because MQCC is a intended to be a replacement for TCP, it must guarantee end-to-end reliability of packets. To accomplish this, we introduce two concepts into MQCC; Random Linear Coding (RLC) and Selective Negative ACKnowledgments (SNACKs).

A. Random Linear Coding (RLC)

We use random linear coding for both error correction and to efficiently utilize multiple paths. Random linear coding is carried out as follows. Data packets arriving at a source node are grouped into generations or blocks, where each generation contains K packets. The source node will form and transmit coded packets, which are random linear combinations of packets from the same generation. The coefficients of each random linear combination are chosen randomly and

uniformly from a large finite field and the coefficients mapping the coded packet to the original data packets are included in the header.

The destination will be able to decode the generation if K or more coded packets arrive, provided that K of the received coded packets are linearly independent. The probability that any K coded packets received by the destination are linearly independent can be made large if the field size is chosen sufficiently large. Here, coding is performed over the finite field GF(256), which is shown in [15] to be sufficiently large in most cases. For more details on the implementation of RLC in MQCC, the reader is referred to our paper [3].

B. Selective Negative Acknowledgments (SNACKS)

While the RLC scheme guarantees that the original packets can be decoded if enough coded packets reach the destination, we still need to explicitly guarantee that enough packets are received. For an efficient method for accomplishing this, we make the following observations about blockage channels.

1) Packet losses are likely to be highly correlated in time.
2) It is likely that some paths will be available and others will not, but less likely that *all* or *none* of the paths will be available (though these last two conditions are indeed possible).
3) Traditional ACK schemes will be inefficient due to the high time-correlation of losses, as well as the difficultly in relaying ACKs along blockage paths.

Based on these observations, we implement a Selective Negative ACKnowledgment (SNACK) reliability scheme. Specifically, the destination has the ability to request coded packets from the source with a negative acknowledgment (NACK). Additionally, because we expect the losses to be correlated in time, we allow the NACK to request a series of coded packets from a series of generations as opposed to a single packet per generation. For example, if a single path out of three is blocked for generations 10 through 20, the destination has the ability to ask for $\frac{K}{3}$ packets from all of the missing generations with a single request, greatly reducing overhead[1]. We refer to these as selective negative acknowledgments (SNACKs). Finally, while the destination will send SNACKs over multiple paths, it is always possible that these packets will also be lost. In our implementation, the destination retries the SNACKs at periodic intervals until it receives enough coded packets to decode the original packets.

V. Performance Evaluation

To validate the performance of MQCC with SNACK-based reliability, we use both simulation and optimization analysis. The simulation tests are done using Riverbed modeler (formerly OPNET modeler) [16] to compare performance of MQCC with TCP Reno with SACK [17]. TCP Reno was chosen as a comparison primarily because it is the default protocol used in the DoD tactical networks we are studying. However, we agree that as the problem with TCP over wireless networks is well known, there are many academic and experimental protocols that have been designed for this specific purpose. While it is possible that some of them may also work significantly better than TCP SACK in a blockage environment, building and testing them all is at best impractical. To compensate for this, we provide analysis on the *optimal* rate allocation for a given blockage network, and show that MQCC performs very close to this optimal solution.

A. Optimization Analysis

To help determine the performance of MQCC, we first define the optimal solution of the problem defined in (4) both with and without blockage.

1) Perfect Channels: We begin with the optimization problem defined in (4) which maximizes the rate utility as constrained by link capacity[2]. We start by calculating the optimal rate allocation for MQCC, which can be defined by the optimization problem

$$\begin{aligned} \text{Maximize} \quad & \sum_{s\in\mathcal{S}} \frac{\gamma}{\rho_s} log(x_s) \\ \text{Subject To:} \quad & \sum_{s\in\mathcal{S}(l)} x_s \le c_l \qquad \forall l \in L \\ & x_s \ge 0 \qquad\qquad \forall s \in \mathcal{S}, \end{aligned} \tag{14}$$

As described in Section III-D, the MQCC algorithm is designed as a distributed solution to this problem, so an initial evaluation of MQCC is whether it solves (14). Calculating the optimal value allows us to calculate the amount of overhead imposed by MQCC, which allows us to determine the cost of the algorithm. These results are presented in Section V-B.

2) Blockage Channels: Next we modify the benchmark formulation to account for blockage channels. We begin by rewriting the optimization problem in (4) as

$$\begin{aligned} \text{Maximize} \quad & \sum_{s\in\mathcal{S}} \frac{\gamma}{\rho_s} \log(r_s) \\ \text{Subject To:} \quad & R\mathbf{r} \le \mathbf{c} \\ & \mathbf{r} \succeq 0, \end{aligned} \tag{15}$$

where $\mathbf{r}$ is the vector of source data rates and $\mathbf{c}$ is the vector of link capacities. R is the routing matrix, and is defined as

$$R_{i,j} = \begin{cases} 1, & \text{if } j \in \mathcal{S}(i) \\ 0, & \text{else.} \end{cases} \tag{16}$$

Blockage will have two effects on the optimization problem in (15). First, if any link on a path is blocked, then the amount of data received along that path will be 0. We represent this by a vector $\mathbf{b} \in \{0,1\}^{|\mathcal{S}|}$ where $\mathbf{b}_s$ is 0 if at least one link used by source s is blocked, and 1 otherwise. Second, we note that source s will not compete for traffic on any link after the

[1] In the case where all paths are blocked, the generations are numbered sequentially allowing the destination to realize that there were losses when the next successful packet is received.

[2] Based on this formulation, the rate is limited by the *lowest* capacity path. This was a design decision of the protocol, and future work will generalize this system to control the rate of each path independently.

first blocked link. We take this into account by defining a time-varying routing matrix that is dependent on the blockage state, defined as

$$R_{i,j}(t) = \begin{cases} 1, & \text{if } j \in \mathcal{S}(i,t) \\ 0, & \text{else.} \end{cases} \tag{17}$$

where $\mathcal{S}(l,t) = \{s \in \mathcal{S} \mid l \in \mathcal{L}(s,t)\}$ and $\mathcal{L}(s,t)$ is the set of links used by source s at time t.

We can then define the benchmark optimization problem for blockage channels as

$$\begin{aligned} \text{Maximize} \quad & \sum_{s \in \mathcal{S}} \frac{\gamma}{\rho_s} log(r_s(t)) \\ \text{Subject To:} \quad & R(t)\mathbf{r}(t) \leq \mathbf{c} \\ & \mathbf{r}(t) \succeq 0, \end{aligned} \tag{18}$$

where $\mathbf{r}(t)$ is the rate vector at time t. We find the throughput from the rate vector using

$$\mathbf{x} = \mathbb{E}[\mathbf{b}(t) \cdot \mathbf{r}(t)]. \tag{19}$$

In practice, we solve (18) iteratively for each change in the channel state. In the next section, we use a simulation implementation of the MQCC protocol using the algorithm defined in (1) to determine how well MQCC performs.

B. Simulation Results

To test the performance of MQCC, we have simulated two test topologies with controlled blockage patterns. Each of these topologies tests a specific performance characteristic. We then implement a realistic topology using measured SATCOM blockage data to compare the performance of MQCC with SACK to existing TCP implementations.

1) Single-Path Simulation: The first topology is shown in Fig. 1, where four flows are defined with a single path for each flow. Flow 0 and Flow 1 are simple single hop flows that are through the satellite link from Node 1 to Nodes 2 and 3 respectively. Flow 2 is from Node 1 to Nodes 5 through the satellite and Node 4, and Flow 3 is from Node 4 to Node 5 directly. The link rates are set at 3 Mbps for the satellite uplink, 5 Mbps for the satellite downlink, and 1.5 Mbps for the terrestrial link. The purpose of this test is to see how the protocol works when the bottleneck links are at different points in the network, and to confirm that the protocol achieves the expected throughput.

First, we test the performance with blockage links. The results for this test are shown in Fig. 2, which shows the throughput for each of the four flows in Fig. 1 for MQCC (in black) and TCP (in red). For comparison, we also show the optimal rate from the calculation in (18) in blue. There are a couple of interesting observations from this plot.

- MQCC is close to optimal for Flows 0 and 1.
- MQCC sends more than the optimal for Flow 2 and significantly less for Flow 3. This is mostly due to queuing at Node 4. Whenever Flow 3 experiences blockage, it will not be transmitting packets from Node 4 to Node 5, allowing Flow 3 to use all of the available capacity on that link. However, since there is a queue at Node 4, there are times when Flow 2 is blocked but *Node 4 still has packets to send*. This is not accounted for in (18), and should slightly increase the throughput on Flow 2 at the expense of Flow 3.
- TCP is able to transmit very few packets over the blockage links. This is not surprising, since TCP will view each blockage event as a buffer overflow congestion event and decrease the throughput accordingly.
- TCP achieves nearly 50% more throughput than MQCC at Flow 3. However, this is due to TCP unfairly sharing capacity between Flow 2 and Flow 3.

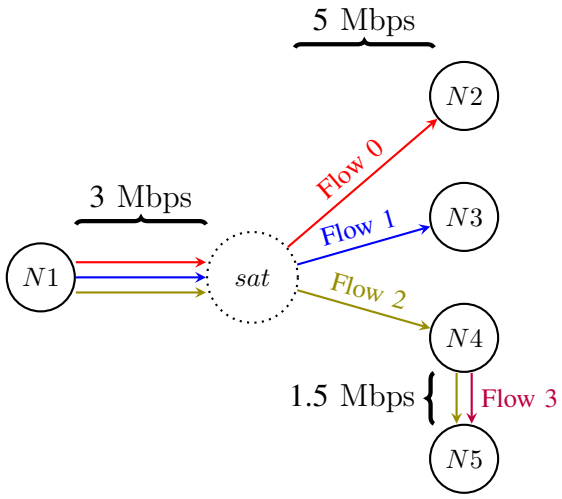

Fig. 1: Topology for single path throughput test.

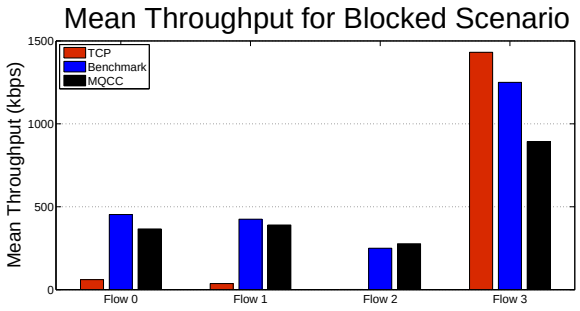

Fig. 2: Throughput of MQCC and TCP assuming blockage channels.

To better understand the performance and fairness of MQCC, we also ran tests with the same topology but with perfect channels. These results are shown in Fig. 3. Again, there are a couple of interesting observations from the figure;

- MQCC transmits slightly less than the optimal rate along all paths. We quantify that gap between optimal and achieved below.
- TCP is very unfair to Flow 2, with Flow 3 achieving nearly 90% of the capacity between Nodes 4 and 5.

To analyze fairness, we used Jain's fairness index [18] to

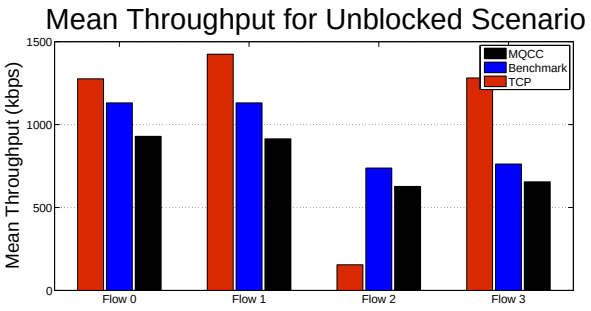

Fig. 3: Throughput of MQCC and TCP assuming perfect channels.

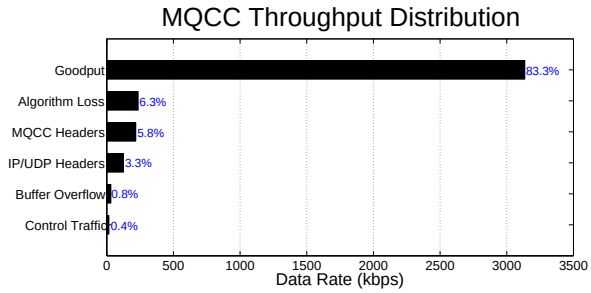

Fig. 4: Throughput distribution for MQCC with perfect channels.

compare the fairness between directly competing flows using MQCC and TCP. For MQCC, we get a fairness rate of 1.00 between Flow 0 and Flow 1, and 1.00 between Flow 2 and Flow 3. For TCP, we instead get 0.97 between Flow 0 and Flow 1, and only 0.62 between Flow 2 and Flow 3. This is because the vast difference in delay between the satellite links and the terrestrial links greatly favors the much faster terrestrial link, resulting in far more packets being sent for Flow 3. Because Flow 3 is sending so few packets, Flow 0 and Flow 1 take advantage of the additional throughput in the satellite link and transmit far more packets. The end result is that, while TCP may achieve a higher overall throughput than MQCC, it is at the cost of fairness in the network.

We also analyzed the difference between MQCC and the optimal benchmark without blockage and present the results in Fig. 4, which breaks down the percentage of bits transmitted by goodput, overhead, and losses. We first note that 83.3% of the transmitted data is relevant data, and the other 16.7% is made up of both overhead and inefficiencies of the protocol. Out of the 16.7%, the two largest components are the MQCC headers and the algorithm losses. While it is beyond the scope of this initial prototype, much of the header data could be compressed resulting in some savings in future versions of MQCC. In contrast, the algorithm losses are due to the rate dynamics of the MQCC congestion control algorithm. While this loss can be decreased, there will be a cost in the percentage of packets lost due to buffer overflow which may outweigh the benefits.

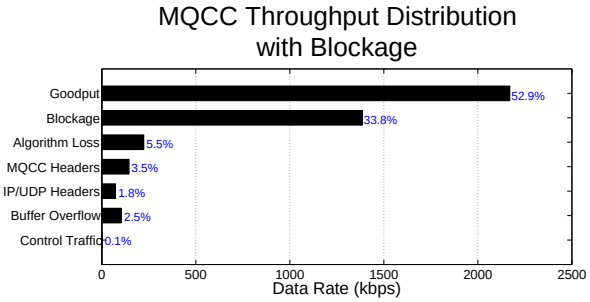

Fig. 5: Throughput distribution for MQCC with blockage channels.

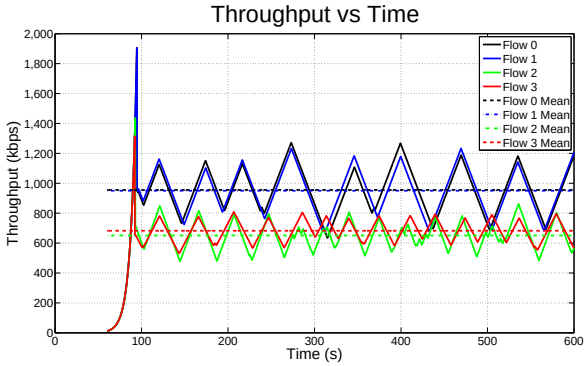

Fig. 6: Throughput with no blockage of the four path topology shown in Fig. 1.

We see a similar pattern in Fig. 5, which shows the deviation from the optimal benchmark when the links have blockage. In this case, the majority of losses are due to the blockage itself, as would be expected. Like the unblocked case, algorithm losses account for a small deviation from the optimal rate. One interesting thing to note is that the number of packets we lose to buffer overflow is much greater with blockage links. This is because blocked links result in delayed queue information at the sending node, resulting in the source not decreasing the rate fast enough.

With the exception of blockage, the primary cause of throughput loss in MQCC is the MQCC algorithm itself, which accounts for 6.3% of the overall throughput in the unblocked case, and 5.5% of the throughput with blockage channels. To help understand the this, we show the transient characteristics of MQCC in Fig. 6. Similar to nearly every congestion control algorithm [4], we see that there is a periodic nature to the rate curves. Flows 0 and 1 oscillate between 635 kbps and 1272 kbps. This large fluctuation is primarily due to the time required to measure reliable information about the bottleneck link (in this case the satellite uplink) which we can not measure directly. Because the blockages can last for many seconds, the algorithm can not increase the data rate too fast, or risk overflowing some links. While it is necessary for this type of blockage network, this large oscillations are the main disadvantage of a protocol like MQCC.

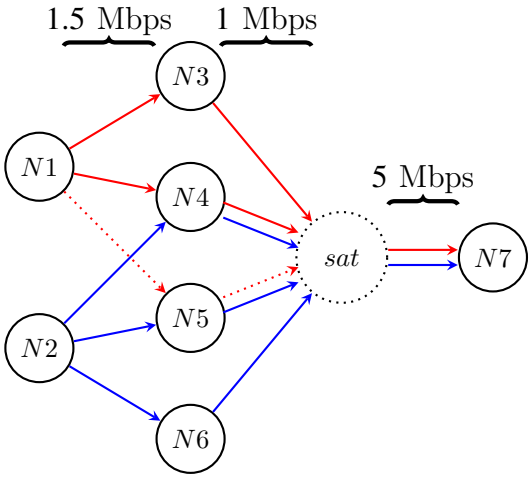

Fig. 7: Two flows, shown both balanced and unbalanced.

2) Multi-Path Simulation: While MQCC will work when restricted to a single path, it is primarily a multi-path protocol. In this section, we describe two test topologies used to evaluate MQCC with multiple paths. We simulate the protocol in each of these topologies and compare the performance to the optimal benchmark.

The first topology, as shown in Fig. 7 (including the dotted paths), has two flows with three paths each. Two of the paths of each flow overlap with that of the other flow. For this topology, the bottleneck is the satellite uplink, which will limit any single path to a maximum of 500 Kbps. Since all paths experience identical conditions, the optimal allocation is to evenly split the capacity between the two flows. The second topology (without including the dotted paths in 7) is an unbalanced topology in which one flow is using three paths, but the second flow is only using two. Again, the bottleneck link is the satellite uplink from Node 4 that is shared by the two flows.

We tested the topology in Fig. 7 both with and without the path shown with the dotted line. in simulation to see whether the two links fairly shared the bandwidth. These results are shown in Fig. 8 and Fig. 9. We see that for both the balanced and unbalanced topologies, the achieved throughput is fair (with Jain's index above 0.99) both with and without blockage.

3) Simulation Demo: Finally, we examine the entire system including the SNACK functionality described in Section IV. For comparison, we are using standard TCP Rome [17] as well as TCP CUBIC [6]. The topology in Fig. 10 shows a single multi-path MQCC flow (designated Flow 0) between the cluster on the left and the cluster on the right. Each cluster is fully connected with 1.5 Mbps transmit rate at each node. Each satellite uplink is 3 Mbps point-to-point, and the downlink is broadcast at 5 Mbps. The flow is divided into three paths, each of which must go through the satellite node. Unlike earlier tests, the OSPF routing protocol between nodes is allowed to attempt to reconverge based on perceived link outages due to blockage. Due to space, we do not show the

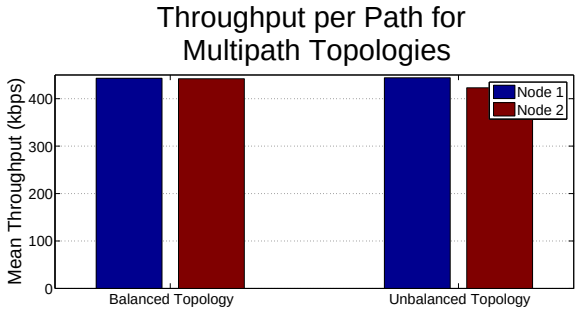

Fig. 8: Throughput of the topologies shown in Fig. 7 with perfect channels.

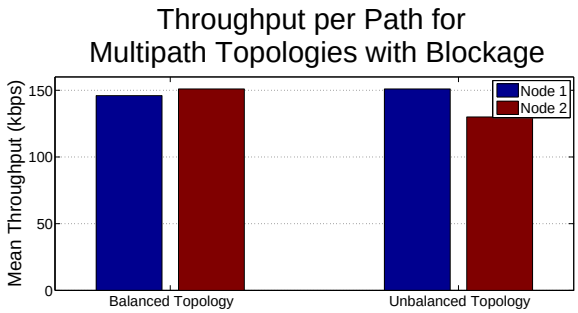

Fig. 9: Throughput of the topologies shown in Fig. 7 with blockage channels.

second flow (Flow 1) that is transmitted from the cluster on the right to the cluster on the left, but it follows a very similar pattern.

We show the throughput obtained by MQCC and both flavors of TCP in Fig. 11. As expected, MQCC achieves far more than either TCP Reno or TCP CUBIC. Indeed, it achieves far more than 3 times the rate achieved by either flavor of TCP, showing that even if we managed to set up three TCP flows over three diverse paths, it would not achieve the throughput obtained by the MQCC flow. In addition, all three results show the throughput with guaranteed reliability. In TCP, this is achieved using Selective ACK, and in MQCC using SNACKs as described in Section IV. While the OSPF reconvergence makes calculating an optimal benchmark for MQCC very difficult, we know that the limiting link in this scenario is the terrestrial link at the source, which is 750 kbps for each path (1.5 Mbps split between the two links). We can see in Fig. 11 that we achieve roughly 60% of that capacity *without taking blockage into account*. The blockage profiles used in this scenario are based on measurements described in [2], and result in $\approx$ 70% availability of at least one of the bottleneck links. While this is clearly only a rough approximation of the optimal benchmark capacity of this system, we see that MQCC is able to achieve very good performance in a scenario where traditional loss-based or delay-based congestion control algorithms can not perform well.

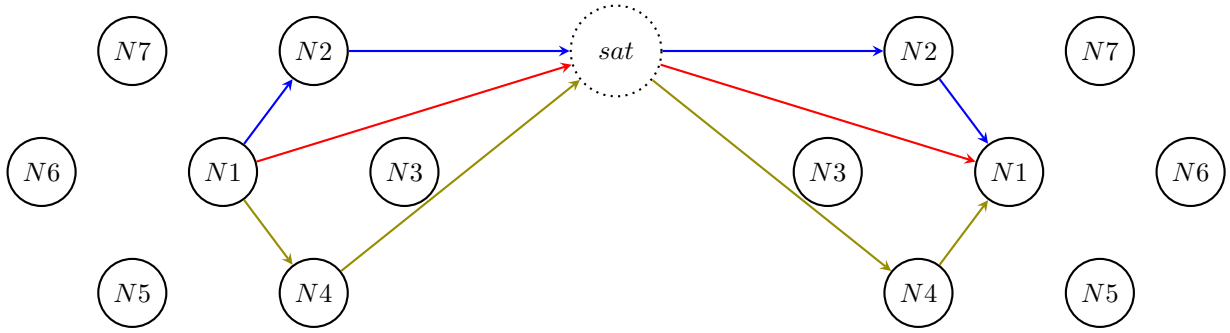

Fig. 10: Topology for system simulation test showing Flow 0.

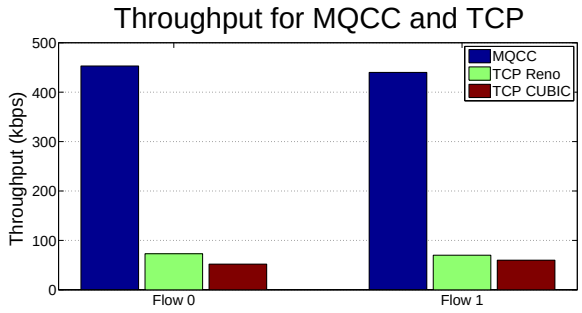

Fig. 11: Throughput with realistic blockage of the two cluster topology shown in Fig. 10.

VI. Conclusions and Future Work

In this work, we build of our other work in [3] and present the Maximum Queue Congestion Control (MQCC) protocol and Selective Negative ACKnowledgment (SNACK) reliability protocol for multipath networks with blockage. We demonstrate that MQCC can be viewed as the distributed solution to a convex optimization problem, and that the explicit queue values can be viewed as the penalty in the optimization solution. We demonstrate the performance of MQCC and SNACKs using network simulation, and show that it performs far better than traditional TCP in terms of fairness, as well as throughput in blockage channels.

We are currently implementing this protocol in an emulation environment using the Extendable Mobile Ad-hoc Network Emulator (EMANE) [19] environment to show that the protocol can work in real time on real hardware. We will then transition the system to work on physical radios and demonstrate that is can work over the air. We are also looking at expanding the congestion indication to be path-specific so the protocol will work multiple paths with disparate data rates. Finally, we are using the ideas from [20] to expand the system to automatically select overlay nodes based on the network topology.

References

[1] "Demo: routing overlay for reliable communication in networks with blockage," in *Proc of ACM Intl. Conf. on Mobile Ad Hoc Networking and Computing (MOBIHOC)*. Philadelphia, PA: ACM, 2014, pp. 73–82.

[2] M. Brennan and W. Smith, "Generalization of Channel Blockage Profiles for SATCOM On-The-Move using 3-D Models," in *Proc. of IEEE Military Communications Conference (MILCOM)*, Washington, D.C., Oct 2006, pp. 1–7.

[3] B. Shrader, S. Pudlewski, L. Herrera, N. M. Jones, , and A. P. Worthen, "Routing Overlay for Reliable Communication in Networks with Blockage," in *Submitted to IEEE Intl. Conf. on Sensor and Ad-hoc Communications and Networks (SECON)*, Seattle, WA, June 2015.

[4] W. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 2011.

[5] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for High-Speed and Long Distance Networks," in *Proc. of IEEE Intl. Conf. on Computer Communications (INFOCOM)*, Barcelona, Spain, April 2006.

[6] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[7] L Brakmo and L Peterson, "Tcp vegas: End to end congestion avoidance on a global internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, 1995.

[8] S. Low, L. Peterson, and L. Wang, "Understanding TCP Vegas: a duality model," *Journal of the ACM (JACM)*, vol. 49, no. 2, pp. 207–235, 2002.

[9] Q. Peng, A. Walid, and S. H. Low, "Multipath TCP Algorithms: Theory and Design," in *Proc. of ACM Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS)*. Pittsburgh, PA: ACM, 2013, pp. 305–316.

[10] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, Implementation and Evaluation of Congestion Control for Multipath TCP," in *Proc. of USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11, Boston, MA, 2011.

[11] J. Sundararajan, D. Shah, M. Medard, M. Mitzenmacher, and J. Barros, "Network coding meets tcp," in *Proc. of IEEE Conf. on Computer Communications (INFOCOM)*, Rio de Janeiro, Brazil, April 2009.

[12] V. Subramanian, S. Kalyanaraman, and K. Ramakrishnan, "An End-to-End Transport Protocol for Extreme Wireless Network Environments," in *Proc. of IEEE Military Communications Conference (MILCOM)*, Washington, D.C., Oct 2006.

[13] S. Pudlewski, T. Melodia, and A. Prasanna, "Compressed-Sensing-Enabled Video Streaming for Wireless Multimedia Sensor Networks," *IEEE Transactions on Mobile Computing*, vol. 11, no. 6, pp. 1060 – 1072, June 2012.

[14] R. Srikant, *The mathematics of Internet congestion control*. Springer, 2004.

[15] P. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. of the annual Allerton conference on communication control and computing*, vol. 41, no. 1, Monticello, Il, 2003, pp. 40–49.

[16] "Riverbed modeler," http://www.riverbed.com.

[17] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options," RFC 2018, October, Tech. Rep., 1996.

[18] R. Jain, D. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," *DEC Research Report TR-301*, September 1984.

[19] "Extendable mobile ad-hoc network emulator (emane)," http://www.nrl.navy.mil/itd/ncs/products/emane.

[20] N. M. Jones, G. S. Paschos, B. Shrader, and E. Modiano, "An overlay architecture for throughput optimal multipath routing," in *Proc of ACM Intl. Conf. on Mobile Ad Hoc Networking and Computing (MOBIHOC)*. Philadelphia, PA: ACM, 2014, pp. 73–82.